Assignment No. 4

```
import time

import random

import threading


class Server:
    def __init__(self, name):
        self.name = name
        self.connections = 0
        self.lock = threading.Lock()  # Added to ensure thread safety

    def handle_request(self):
        with self.lock:  # Ensures connections are updated safely
            self.connections += 1

        time.sleep(random.uniform(0.5, 2))  # Simulate request processing

        with self.lock:
            self.connections -= 1

class LoadBalancer:
    def __init__(self, servers):
        self.servers = servers

class RoundRobin(LoadBalancer):
    def __init__(self, servers):
        super().__init__(servers)
        self.index = 0
```

```
        self.lock = threading.Lock()  # Ensure thread safety for index update

    def distribute(self, req_id):
        with self.lock:
            server = self.servers[self.index]
            self.index = (self.index + 1) % len(self.servers)

        print(f"Request {req_id} handled by {server.name} (Round Robin)")
        server.handle_request()

class LeastConnections(LoadBalancer):
    def __init__(self, servers):
        super().__init__(servers)
        self.lock = threading.Lock()  # Ensure thread safety for server selection

    def distribute(self, req_id):
        with self.lock:
            server = min(self.servers, key=lambda s: s.connections)

        print(f"Request {req_id} handled by {server.name} (Least Connections)")
        server.handle_request()

def client_request(req_id, lb):
    lb.distribute(req_id)

def main():
    servers = [Server(f"Server-{i}") for i in range(3)]
    lb = RoundRobin(servers)  # Or use LeastConnections(servers)
```

```
    threads = []
    for req_id in range(1, 11):
        thread = threading.Thread(target=client_request, args=(req_id, lb))
        thread.start()
        threads.append(thread)
        time.sleep(random.uniform(0.1, 0.5))  # Simulating random request intervals

    for thread in threads:
        thread.join()  # Ensures all threads finish before exiting

if __name__ == "__main__":
    main()
```

Output: